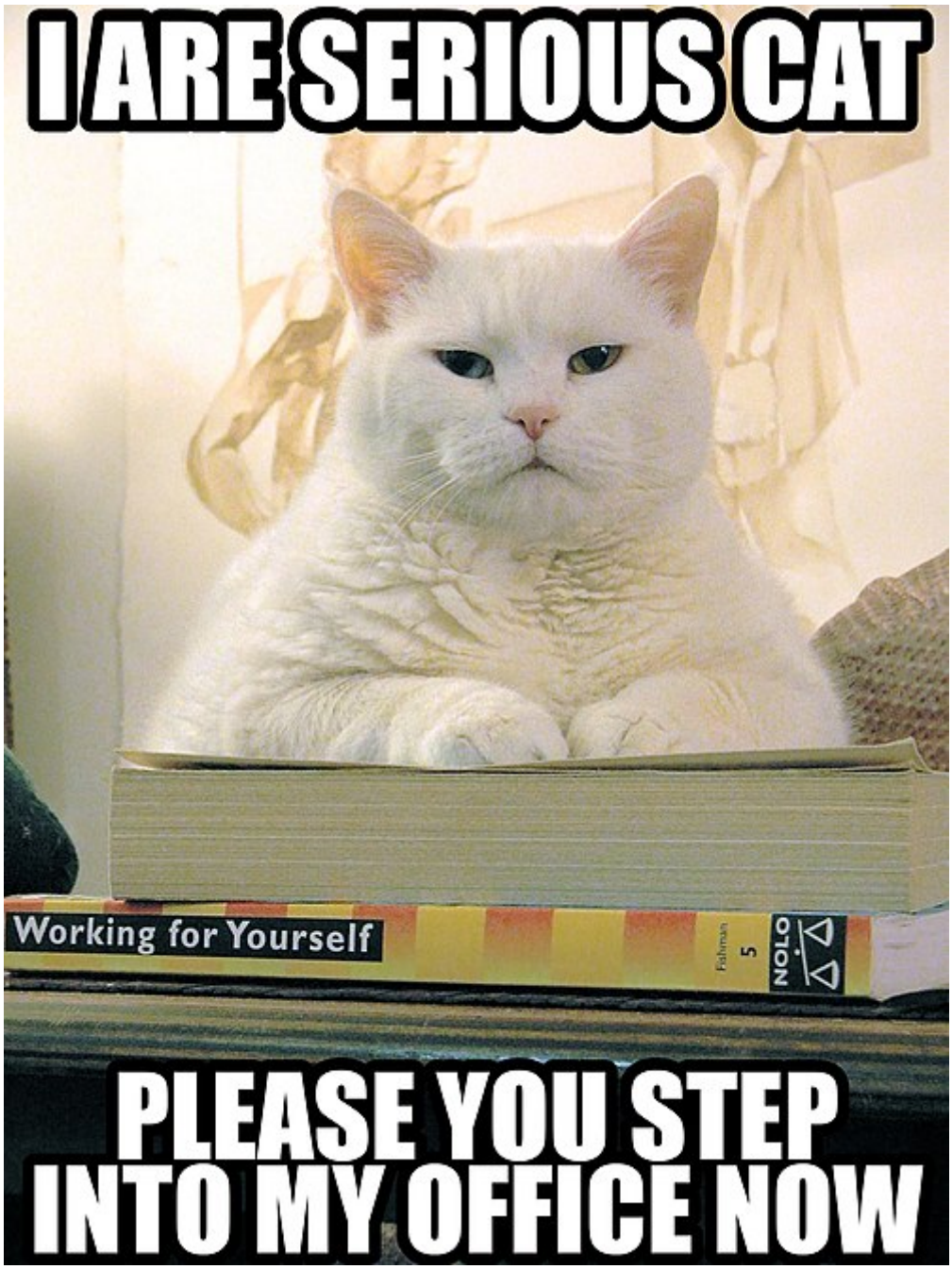
Programming in LOLGraphics 3.4/Printable version

Programming in LOLGraphics 3.4

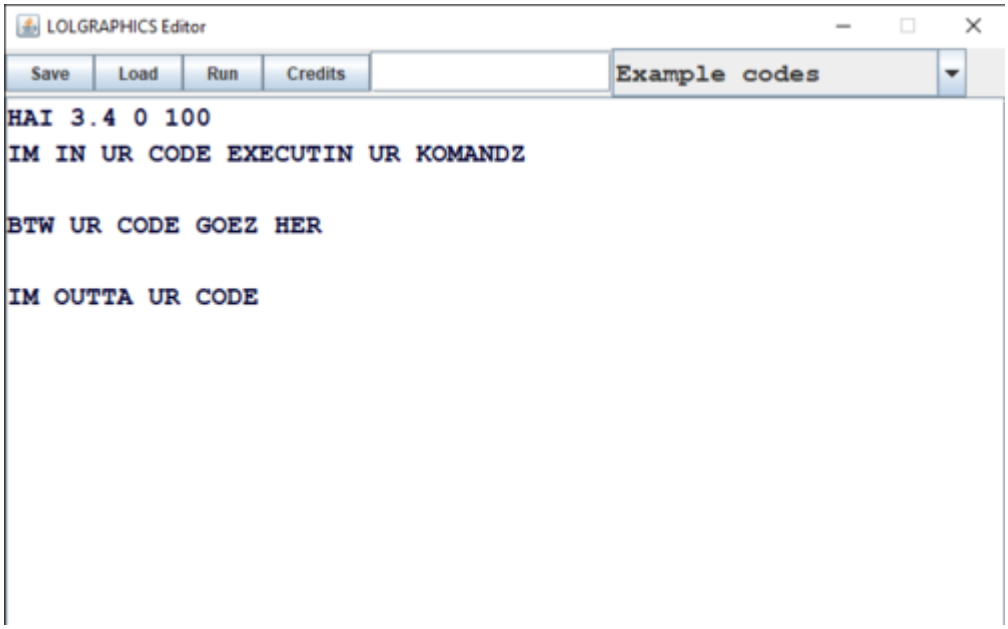
Preface

LOLGraphics is an esoteric programming language created and published by Bloody Wall Software. It was inspired by LOLCODE but other than the spelling mistakes the two languages bear little resemblance. Some of the features are loosely based on Assembly Language but always with unique twists.



Interface, hello world program, subprograms, & labels

Editor Interface



When opening LOLGraphics, you will be greeted to the LOLGraphics Editor. Let's go over what each button does:

The save button saves the current code to example.lulz in the same folder that the jar file of LOLGraphics is located in. The load button reads example.lulz and displays it's contents on the editor. This is assuming you wrote in the file field the word "example". Otherwise, it will read and write to whatever file name you wrote there with a .lulz extension.

The run button takes what is written in the editor, removes all blank lines and comments (a line that begins with BTW, or a line that begins with OBTW all the way till the line TLDR) and trims all the lines (removes whitespace characters from the start and end of it). Then it will send what's left to the interpreter to run it line by line.

Interpreter interface

Here's how the interface of the interpreter works:

The screen of the interpreter consists of 3 main parts: a console for text output, a field for input, and a graphics area. Don't worry if you don't understand how to use them just yet as all of it will covered later on in the book.

You can check out the example codes available in the editor. Just take into account that not all of them use the latest version of LOLGraphics. This book will only cover version 3.4 which is the latest.

Code Structure

When you open the editor, you will see the default code which looks like that:

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

BTW UR CODE GOEZ HER

IM OUTTA UR CODE
```

Let's go over what each line does. The first line might seem useless for the untrained eye but it actually has multiple purposes. The first is to specify the version of LOLGraphics you are using. Believe it or not but if you download LOLGraphics Editor x.y, you are actually downloading all the versions up to this version. That means that a code written for LOLGraphics 0.1 will work fine on all versions of the editor, and will load using LOLGraphics Interpreter 0.1. The two numbers represent the delays between two commands. In this example 0 milliseconds since the interpreter I launched till the first command starts, and 100 milliseconds (0.1 second) in between every 2 commands. If you write just one number it will be the delay between every 2 commands and the initial delay will be 0. If you don't write any numbers it will run as if you wrote 0 and 100.

This is also why throughout this book, the efficiency of the code will be measured purely by the number of lines executed by the interpreter.

Also it's possible to change the delay in the middle of the code using the command `PLZ SET DELAY <number>`.

The line `IM IN UR CODE EXECUTIN UR KOMANDZ` does NOTHING🎉🎉🎉 but is necessary for the code to run. Otherwise you will get a message saying that you forgot to specify where the code starts. Remember, this is a meme programming language.

Finally the line `IM OUTTA UR CODE` end the code. There's no limitation on where you can put, or how many times you can put it. It's just a command that ends the code and can be placed anywhere. You can theoretically omit the line and it will run until it has executed the last line. However, if you want to have subprograms it's a must to have this line otherwise the interpreter will run on all the subprograms.

Error handling

LOLGraphics forces you to be polite and most (not all) commands begin with PLZ. If you type a command that doesn't exist it will check if it begins with PLZ. If yes it will tell that it doesn't know what to do and ask to give a command that actually exists. Otherwise it will yell at you that you forgot the magic word. Don't worry about the wording to much and definitely don't add PLZ to commands that don't have it otherwise it won't work.

When you get yelled at, the interpreter will not tell you what line to fix so it's recommended that when writing your code, you write it in small parts and test each one before you continue to the next.

Printing strings

There are two ways to print strings in LOLGraphics:

```
PLZ TYPE TEXT HELLO WORLD
PLZ PRINT TEXT HELLO WORLD
```

The first one prints Hello World to the console, and the second one prints hello world and then goes to the next line. If you want to print variable x (variables will be taught in a later chapter), as x: 5, adding a space at the end of the type command won't help since the LOLGraphics Editor deleted spaces at the beginning and end of the line. To compensate there is a command `PLZ ADD A SPACE` that types that character to the screen.

Hello World

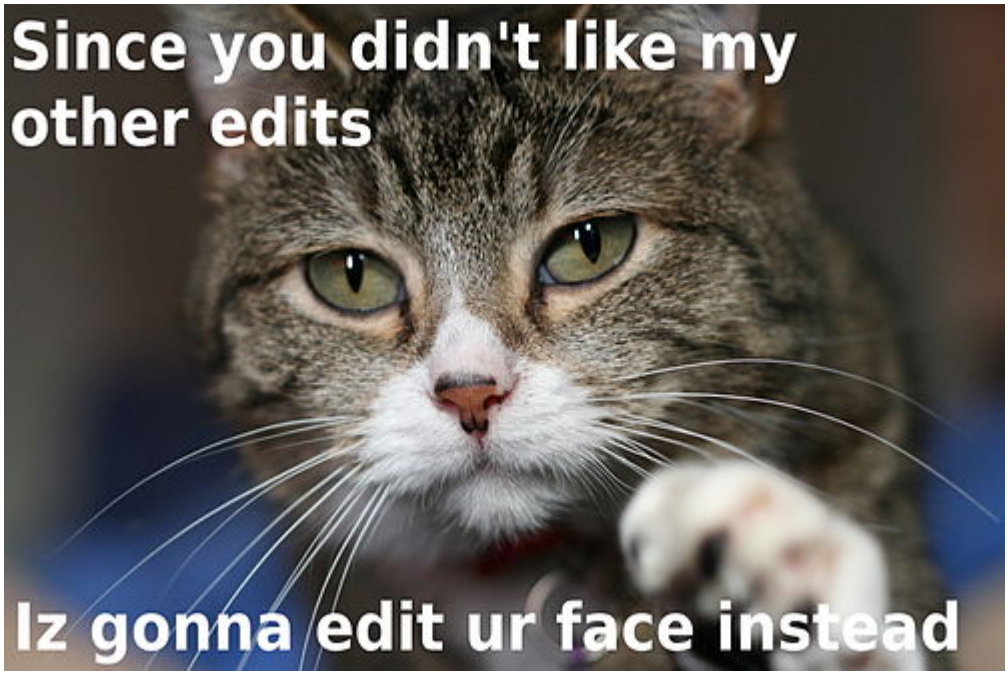
This is how a hello world program in LOLGraphics looks like:

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ PRINT TEXT HAI WORLD!

IM OUTTA UR CODE
```

Clearing the console & changing text color



You can clear the console using `PLZ CLEAR TEH CONSOLE`.

You can also change the text color using the command `PLZ CHANGE TEXT COLOR <color>`. There are 3 ways to specify the color:

- Type it's name for the following colors: black, blue, cyan, dark gray, gray, green, light gray, magenta, orange, pink, red, white, and yellow.

```
PLZ CHANGE TEXT COLOR BLUE
```

- Type it's RGB values.

```
PLZ CHANGE TEXT COLOR 255 0 0
```

- Type the word random for a random color.

```
PLZ CHANGE TEXT COLOR RANDOM
```

Subprograms

A subprogram begins with `IM IN UR SUBPROGRAM DAT IZ KALLED <name>`. In order to end the subprogram, you type `IM OUTTA UR SUBPROGRAM` which like the command to end the code, can be placed anywhere in the subprogram. After the subprogram ends it will continue running starting from the line after where the subprogram was called. In later chapters, you will learn how to use subprograms in loops and conditions. You can also call a subprogram inside a subprogram and create recursion!🎊 Heres a code that prints `I can has a ceezburger?` indefinitely using recurion.

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM PRINT

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED PRINT
PLZ PRINT TEXT I CAN HAS A CHEEZBURGER?
PLZ RUN SUBPROGRAM PRINT
```

In this code there's no point writing `IM OUTTA UR SUBPROGRAM` since it will never get there anyways. That's another thing about LOLGraphics, you can add commands between the "main area" and the first subprogram or in between 2 subprograms but that probably isn't very smart since the interpreter will never run them. Also they can be used as comments since they won't be run anyways but it's probably smarter to write regular comments with BTW or OBTW/TLDR since they will be wasted before the code is sent to the interpreter.

Labels

When you start a subprogram, the program remembers which line it was that called the subprogram so that it can go back and continue running form that line. Usually that's useful, but you might not always want that. Take for example the recursive code above. There's no any reason for the code to remember where this subprogram was called. That's why labels exist. To define a label, you write `DIS IZ MY LABEL! IT IZ KALLED <name>` (and yes, the exclamation mark is mandatory) and to jump to a label, write `PLZ GOTO LABEL label`. Now let's rewrite the infinite recursion above only with labels.

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

DIS IZ MY LABEL! IT IZ KALLED L
PLZ PRINT TEXT I CAN HAS A CHEEZBURGER?
PLZ GOTO LABEL L

IM OUTTA UR CODE
```

This code does exactly the same. The only differences are that it's shorter and more efficient.

Variables & Input

LOLGraphics Memory

In LOLGraphics, the interpreter gives you access to 4 segments which can be used to store data. Each segment has 65536 cells. In the one byte segment, each cell is a signed 1 byte integer. In the two byte segment, each cell is signed 2 byte integer. In the four byte segment each cell is a signed 4 byte integer, and in the 8 byte segment each cell is a signed 8 byte integer.

The cells in each segment are numbered 0-65535. At the beginning of the program each cell is set to a random number. You can print the value of each cell using `PLZ PRINT ONE/TWO/FOUR/EIGHT BYTE 100`. That command will print the value in the cell with index 100 (the 101st cell!) and not the number 100 (!). If you want to print 100 regardless of the value of a variable, simply type `PLZ PRINT TEXT 100`. can also change the value of each cell. Running `PLZ SET ONE BYTE 100 27` will set the value on index 100 in the one-byte segment to 27.

Declaring a variable

Writing cell addresses will make your code hard to read if you use many values. That's why you have the ability to label those cells. If you type the line `I HAS A FOUR BYTE DAT IZ CALLED X`, you label the first unlabeled cell in the four byte segment "X". You can also run `PLZ SET FOUR BYTE X 27`, and `PLZ PRINT FOUR BYTE X`.

The following code will print the same number twice but each time a different one.

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A FOUR BYTE DAT IZ CALLED X
PLZ PRINT FOUR BYTE X
PLZ PRINT FOUR BYTE 0

IM OUTTA UR CODE
```

Variable names

Readers familiar with programming languages will definitely recall that practically all languages have rules regarding to which characters can be used in the name of a variable (usually just letters, numbers, and underscores with the first character being a letter always). The esoteric programming language TMMLPTEALPAITAFNFAL (The Multi-Million Language Project To End All Language Projects And Isn't That A Fine Name For A Language) has really weird rules that rotate every day so one day it can require the symbol [to appear at least 15 days or something like that.

LOLGraphics doesn't have any restrictions regarding the variable name. You can name a variable by a number but this probably isn't a very smart decision since it will always go the cell with that index and not that variable.

Also you can give different variables the same name. If they are in different segments then there isn't any reason the code won't work as you have to specify the size of the variable anyways. However, if they are in the same segment the code will still work but when the variable name is specified, it will always operate using the first instance of it.

Where to declare variables

Many programming languages like java and rust have scopes and variables declared in a scope can't be used outside of it. LOLGraphics doesn't have such stuff so if you declare a variable it's forever. The only important thing is that the command which declares the variable was run before the commands that use it are run. In order to avoid having multiple variables with the same name, it's recommended to declare the variables either at the very beginning of the code or if you think it makes the code messy, in a subprogram that is called only at the beginning of the code.

Also, it's important to remember that despite the fact that this books says "and address or a variable" that a variable in LOLGraphics is just a label that points towards an address.

Random numbers

To generate a random number, all you need to do is type the command `PLZ GIMME A RANDOM ONE/TWO/FOUR/EIGHT BYTE <address/variable>`. There is also the command `PLZ CLEAR ALL TEH SEGMENTS`. Despite what the name suggests, it doesn't set all the cells in all the segments to 0, and neither does it set them to their initial values. Instead it sets all the values in all the segments to random numbers.

Input

There are 2 ways to ask the user for input. The first one, which is preferable for cli apps is using the command `PLZ ASK TEH USR 2 GIMME A ONE/TWO/FOUR/EIGHT BYTE <address/variable>` this command will stop the program, activate the input button, wait for the user to type a number, then after he presses the button, it will save the value typed by the user in the specified address or variable.

The second way which is preferable in some situations is using the command `PLZ READ ONE/TWO/FOUR/EIGHT BYTE <address/variable>`. It doesn't wait for the user to press on the input button, and doesn't even activate it, instead it just takes whatever is there, and puts in the specified variable or address. If the input field is empty, the command will do nothing. This is the method used in the "8462 demo" which is like a WASD demo only using the numpad. You will also learn how to write such a program later in the book.

Halting the program

The above section is useful if you want to wait for the user to type a number and then press on the input button. But what if you want to just halt the program and wait for the user to press on the input button? Luckily there is a command just for that! It's `PLZ WAIT 4 DA USR 2 REACT`. It will be annoying for production applications if you plan on writing any in LOLGraphics, however it will be useful for debugging purposes.

Conditions & Loops

Introduction

What we have learned so far is very good, but in order to write anything useful in a programming langage, it's a must to have conditions and loops or at least a way to make "pseudo-loops" using conditions and labels. Luckily, LOLGraphics has all 3 and in this chapter we will be learning all of them! 🎊

If else statements in other programming languages

In python for example, an if else statement looks the following:

```
if x==5:
   #Instructions
elif x=6:
   #Instructions
else:
   #Instructions
```

Different programming languages may have different syntaxes, but the overall idea stays the same. For example in java you must put the conditions in brackets, write else if instead of elif, and if the block is more than one line wrap it with curly brackets {}.

However, in LOLGraphics it looks entirely different.

If else statements in LOLGraphics

In LOLGraphics there is a single flag that can be updated using commands and is used in conditions and loops. It's default value is false, and to update it type `PLZ ASK CEILIN KAT 2 CHEK IZ <condition>`. The condition can be comparing between 2 numbers, addresses, or variables, or a combination of two.

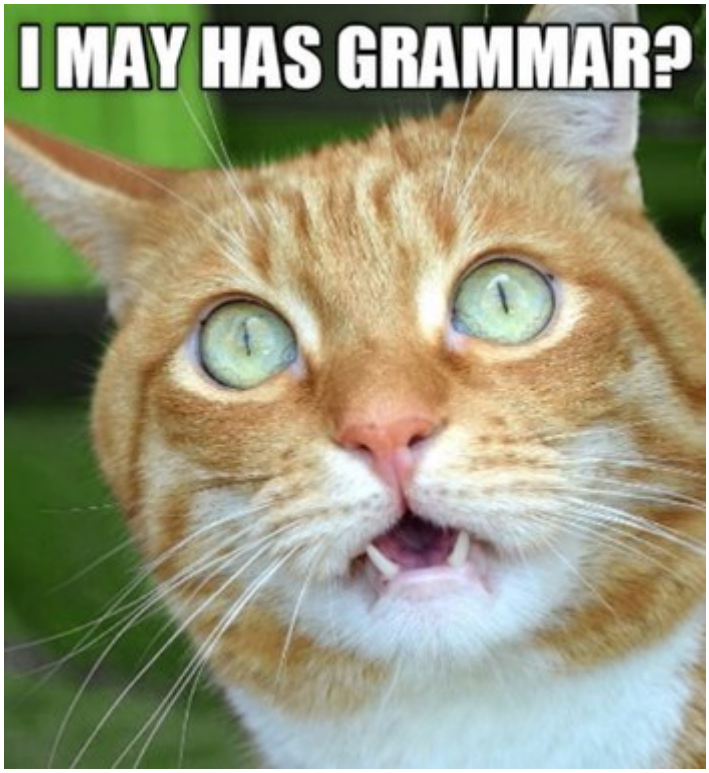
The command `PLZ ASK CEILIN KAT 2 CHEK IZ [X]==8` will check is the value in the one byte variable X is equal to 8. You can use two byte variables ([[X]]), four byte variables ([[[X]]]), and eight byte variables ([[[[X]]]]).

So we know how to set the flag, but what to do with it. Luckily for use, ceiling cat is very generous and will keep nodding while the flag is true. Simply type `IF CEILIN KAT IZ NODDING PLZ RUN <subprogram>`. If you want to run a subprogram when the flag is false, type `ELSE PLZ RUN <subprogram>`. Note that unlike other programming languages, those are two separate commands that come independently from each other which means that you can have any "else" without an "if" in LOLGraphics.

Also, you can manually set the flag to true and false by using the commands `PLZ ASK CEILIN CAT 2 NOD` and `PLZ ASK CEILIN CAT 2 STOP NODING`

Finite Recursion & Roman Numerals

In the previous chapter, you have learned how to make infinite recursions. Using the if/else statement, you can make a finite recursion. Using this you can write a program that coverts numbers to Roman Numerals. Here's the source code that is not included as an example program in the editor:




```
HAI 3.4 0 10
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A TWO BYTE DAT IZ CALLED X
DIS IZ MY LABEL! IT IZ KALLED START
PLZ ASK TEH USR 2 GIMME A TWO BYTE X
PLZ PRINT TWO BYTE X

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>999
IF CEILIN KAT IZ NODDING PLZ RUN M

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>899
IF CEILIN KAT IZ NODDING PLZ RUN CM

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>499
IF CEILIN KAT IZ NODDING PLZ RUN D

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>399
IF CEILIN KAT IZ NODDING PLZ RUN CD

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>99
IF CEILIN KAT IZ NODDING PLZ RUN C

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>89
IF CEILIN KAT IZ NODDING PLZ RUN XC

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>49
IF CEILIN KAT IZ NODDING PLZ RUN L

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>39
IF CEILIN KAT IZ NODDING PLZ RUN XL

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>9
IF CEILIN KAT IZ NODDING PLZ RUN X

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>8
IF CEILIN KAT IZ NODDING PLZ RUN IX

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>4
IF CEILIN KAT IZ NODDING PLZ RUN V

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>3
IF CEILIN KAT IZ NODDING PLZ RUN IV

PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>0
IF CEILIN KAT IZ NODDING PLZ RUN I

PLZ PRINT TEXT
PLZ PRINT TEXT======================
PLZ GOTO LABEL START

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED M
PLZ TYPE TEXT M
PLZ SET TWO BYTE X X-1000
PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>999
IF CEILIN KAT IZ NODDING PLZ RUN M
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CM
PLZ TYPE TEXT CM
PLZ SET TWO BYTE X X-900
```

```
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED D
PLZ TYPE TEXT D
PLZ SET TWO BYTE X X-500
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CD
PLZ TYPE TEXT CD
PLZ SET TWO BYTE X X-400
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED C
PLZ TYPE TEXT C
PLZ SET TWO BYTE X X-100
PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>99
IF CEILIN KAT IZ NODDING PLZ RUN C
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED XC
PLZ TYPE TEXT XC
PLZ SET TWO BYTE X X-90
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED L
PLZ TYPE TEXT L
PLZ SET TWO BYTE X X-50
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED XL
PLZ TYPE TEXT XL
PLZ SET TWO BYTE X X-40
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED X
PLZ TYPE TEXT X
PLZ SET TWO BYTE X X-10
PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>9
IF CEILIN KAT IZ NODDING PLZ RUN X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED IX
PLZ TYPE TEXT IX
PLZ SET TWO BYTE X X-9
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED V
PLZ TYPE TEXT V
PLZ SET TWO BYTE X X-5
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED IV
PLZ TYPE TEXT IV
PLZ SET TWO BYTE X X-4
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED I
PLZ TYPE TEXT I
PLZ SET TWO BYTE X X-1
PLZ ASK CEILIN KAT 2 CHEK IZ [[X]]>0
IF CEILIN KAT IZ NODDING PLZ RUN I
IM OUTTA UR SUBPROGRAM
```

Notice how some of the digits are checked recursively, while others like L for 50, XL for 40, and IV for 4 can appear only once and therefore are only printed once.

“8462 demo” part 1

Remember previously, the “8462 demo” was mentioned? It’s like a WASD demo only using the numpad. Let’s implement part of it here. Note that this is only part of the code, so don’t expect to copy it to the editor and have it working. Also don’t worry too much about what exactly the subprograms do, we will cover that in the chapter about graphics. Concentrate on the conditions. Assume that the variable graphics was given by the user previously in the code.

```
PLZ ASK CEILIN KAT 2 CHEK IZ [BUTTON]==8
IF CEILIN KAT IZ NODDING PLZ RUN UP

PLZ ASK CEILIN KAT 2 CHEK IZ [BUTTON]==2
IF CEILIN KAT IZ NODDING PLZ RUN DOWN

PLZ ASK CEILIN KAT 2 CHEK IZ [BUTTON]==4
IF CEILIN KAT IZ NODDING PLZ RUN LEFT

PLZ ASK CEILIN KAT 2 CHEK IZ [BUTTON]==6
IF CEILIN KAT IZ NODDING PLZ RUN RIGHT
```

As you can see the code is neither attractive nor efficient. Later this chapter we will learn how to write a better version of this code. Also, the code above contains 8 lines of code since empty lines and comments are not sent to the interpreter.

Ceiling Cat

Switch case

Luckily, there is a way to make the code in the previous section more efficient & more attractive, using the switch case commands. The LOLGraphics Interpreter stores a single 8 byte variable (it’s the largest size possible in LOLGraphics in order that you will be able to put in it also smaller sizes). It can be updated using the SWITCH command, and checked using the CASE command. Let’s use a more efficient version of the code in the previous section as an example:

```
SWITCH [BUTTON]
CASE 8 UP
CASE 2 DOWN
```

```
CASE 4 LEFT
CASE 6 RIGHT
CASE 86,68,9 UP_RIGHT
CASE 48,84,7 UP_LEFT
CASE 42,24,1 DOWN_LEFT
CASE 62,26,3 DOWN_RIGHT
```

The first line takes whatever is in the one byte variable BUTTON and puts it in the switch/case variable (the default value is 0). The second line of code checks is the variable equal to 8, and if it is goes to the subprogram called UP (again, don't worry to much about what it does, we will learn more about that in the chapter about graphics).

The third, fourth, and fifth do more or less the same, so let's skip them and explain the sixth line of code. This line has multiple numbers separated by commas without spaces. This is the way it must be done. If you don't type commas or do type a space between values, your code won't work! If the value of the switch/case variable is equal to 86, 68, or 9, subprogram UP_RIGHT will run. If you check the numpad, it will make sense.

The code above consists of 9 lines of code, just one more than the code in the previous section however accomplished much more since it allows also to move by diagonals. If you wanted to make it only up/down/left/right, it would be 5 lines of code.

If statements or switch/case?

As you have seen the code using switch case is both cleaner and more efficient. So why on earth would you want to use if/else? In this specific program, using switch/case is preferable, however you should know that it's less flexible since it can only be used to check is a variable equal to a constant number. If you want to check is a variable equal to another variable or use more less operations, then you must use if else. Also sometimes you want to compare a variable to a variable, and also in this case switch/case won't be very useful for you.

Infinite loops

To create an infinite loop, write `FOREVER RUN subprogram`. Now let's rewrite the infinite loop code from "Interface, hello world program, subprograms, & labels", only with actual loops:

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

FOREVER RUN PRINT

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED PRINT
PLZ PRINT TEXT I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM
```

Finite loops

Loops work sort of like if statements in that that you can't write the condition itself in the same line. The loop checks is ceiling cat nodding and while it is, runs a subprogram. The command looks as following: `WHILE CEILIN CAT IZ NODDIN PLZ RUN <subprogram>`. If you want it to work as an

actual while loop, update the flag before this line and at the end of the subprogram.

Factorial

The following code prints the factorial of a number the user enters. It is not available as an example code in the LOLGraphics Editor.

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A EIGHT BYTE DAT IZ CALLED X
I HAS A EIGHT BYTE DAT IZ CALLED I
PLZ ASK TEH USR 2 GIMME A EIGHT BYTE X
PLZ SET EIGHT BYTE I X-1

PLZ ASK CEILIN KAT 2 CHEK IZ [[[[I]]]]>1
WHILE CEILIN CAT IZ NODDIN PLZ RUN MULTIPLY
PLZ PRINT EIGHT BYTE X

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED MULTIPLY
PLZ SET EIGHT BYTE X X*I
PLZ SET EIGHT BYTE I I-1
PLZ ASK CEILIN KAT 2 CHEK IZ [[[[I]]]]>1
IM OUTTA UR SUBPROGRAM
```

Fibonacci sequence

The following code prints numbers from Fibonacci sequence. It is available in the LOLGraphics Editor as one of the example codes.

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A ONE BYTE DAT IZ CALLED MAX
PLZ TYPE TEXT HOW MANY NUMBERS DO   YOU WANT ME TO PRINT?
PLZ ASK TEH USR 2 GIMME A ONE BYTE MAX
PLZ PRINT ONE BYTE MAX
PLZ PRINT TEXT ======================
```

```
I HAS A ONE BYTE DAT IZ CALLED I
I HAS A EIGHT BYTE DAT IZ CALLED A
I HAS A EIGHT BYTE DAT IZ CALLED B
I HAS A EIGHT BYTE DAT IZ CALLED C

PLZ SET ONE BYTE I 2
PLZ SET EIGHT BYTE A 0
PLZ SET EIGHT BYTE B 1
PLZ ASK CEILIN KAT 2 CHEK IZ [MAX]>0
IF CEILIN KAT IZ NODDING PLZ RUN PRINT_A
PLZ ASK CEILIN KAT 2 CHEK IZ [MAX]>1
IF CEILIN KAT IZ NODDING PLZ RUN PRINT_B
PLZ ASK CEILIN KAT 2 CHEK IZ [I]<[MAX]
WHILE CEILIN CAT IZ NODDIN PLZ RUN COUNT

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED COUNT
PLZ SET ONE BYTE I I+1
PLZ SET EIGHT BYTE C A
PLZ SET EIGHT BYTE A B
PLZ SET EIGHT BYTE B C+A
PLZ PRINT EIGHT BYTE B
PLZ ASK CEILIN KAT 2 CHEK IZ [I]<[MAX]
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED PRINT_A
PLZ PRINT EIGHT BYTE A
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED PRINT_B
PLZ PRINT EIGHT BYTE B
IM OUTTA UR SUBPROGRAM
```

Graphics in LOLGraphics

Introduction

So, in the previous chapter you have learned how to make cli programs in LOLGraphics, and hopefully have gotten an appreciation of where the first 3 letters in the name of the programming language have come from. In this chapter you will learn the graphics portion of LOLGraphics, and in the next chapter, we will combine everything we have learned and make a simple game in LOLGraphics.

P.S. The “8462 demo“ will be in this chapter, the game in the next chapter will be more advanced.

Drawing a cheeseburger

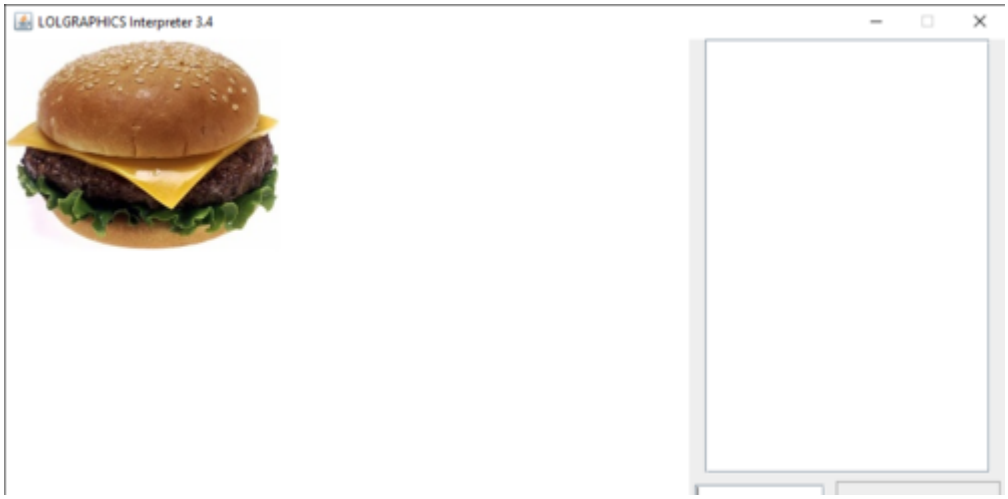
We shouldn’t forget that this programming language was inspired by LOLCODE which in turn was inspired by the LOLCAT memes. One of the most famous of them is a cat with the text “I Can Has Cheezburger?” (lolspeak for Can I have a cheeseburger). Therefore, it only makes sense to include in LOLGraphics a command that draws a cheeseburger on the screen. If you write the following code, you will see a cheeseburger 🍔 on the top left of the screen:

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

I CAN HAS A CHEEZBURGER?

IM OUTTA UR CODE
```

And here’s the output of this program:



Now this is all interesting, but what if we want to draw cheeseburgers in different places in the screen, not just the top left?

Coordinates in LOLGraphics

In order to customize the location of the cheeseburger, we need to use LOLGraphics coordinates system, so first we need to understand how it works. It will be intuitive for people who have worked with graphics in languages other than scratch, but not for others.

The top left comer of the screen is the point (0,0) aka x=0, and y=0. The positive direction of the x axis is right, and the positive direction of the y axis is down. The negative axis of the x axis is left, and the negative axis of the y axis is up.

This might be confusing for people not accustomed with programming that are used from Mathematics that the positive direction of the y axis is up. However, in computers it's down.

Customizing the location of the cheeseburger

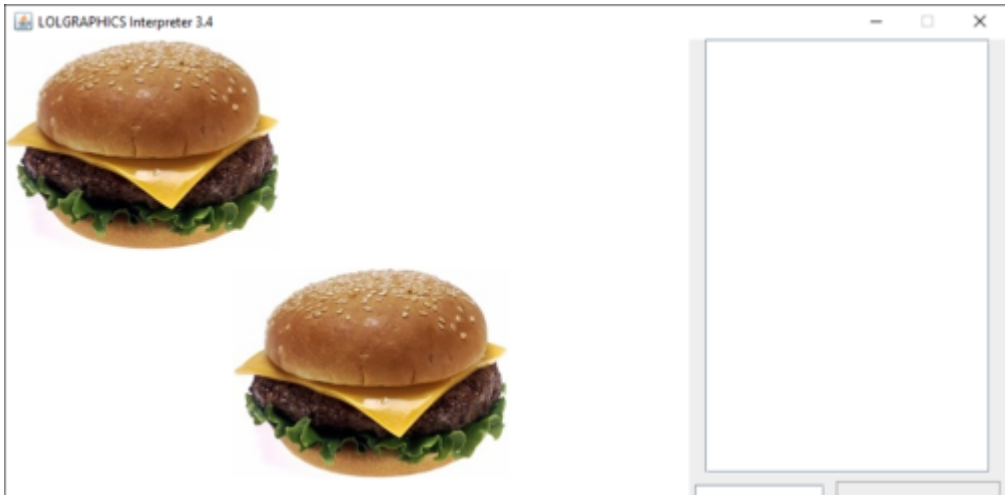
The command `I CAN HAS A CHEEZBURGER?` does not accept any parameters. How then are we accepted to tell the interpreter where we want to draw the cheeseburgers? Using the command `PLZ DELIVR MAH CHEEZBURGERS 2 <x> <y>`! This command doesn't affect the status of cheeseburgers already drawn on the screen (there can be as many cheeseburgers as you like) but only cheeseburgers that will be drawn after it. The command can accept as arguments numbers or two-byte variables.

The following code will draw 2 cheeseburgers on the screen, one on the top left, and one on the coordinates (200,200):

```
HAI 3.4 0 100
IM IN UR CODE EXECUTIN UR KOMANDZ

I CAN HAS A CHEEZBURGER?
PLZ DELIVR MAH CHEEZBURGERS 2 200 200
I CAN HAS A CHEEZBURGER?

IM OUTTA UR CODE
```



Now sometimes you might want to clear what was drawn on the screen and start from scratch. Thats why the command `PLZ CLEAR TEH SCREEN` exists.

8462 demo

The 8462 demo is similar to a WASD demo, only using the numpad. It is included in the editor as an example program. Now we have learned all the things necessary to make it. Here's the source code:

```
HAI 2.5 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A ONE BYTE DAT IZ CALLED BUTTON

PLZ SET TWO BYTE X 150
PLZ SET TWO BYTE Y 100

PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
FOREVER RUN CHECK

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED CHECK
PLZ SET ONE BYTE BUTTON -1
PLZ READ ONE BYTE BUTTON

SWITCH [BUTTON]
CASE 8 UP
CASE 2 DOWN
CASE 4 LEFT
CASE 6 RIGHT
CASE 86,68,9 UP_RIGHT
CASE 48,84,7 UP_LEFT
CASE 42,24,1 DOWN_LEFT
CASE 62,26,3 DOWN_RIGHT

IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ CLEAR TEH SCREEN
```

```
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP_RIGHT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X+10
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP_LEFT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X-10
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN_RIGHT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X+10
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN_LEFT
PLZ CLEAR TEH SCREEN
PLZ SET TWO BYTE X X-10
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
I CAN HAS A CHEEZBURGER?
IM OUTTA UR SUBPROGRAM
```

Scrolling background

Using what you learned above it's possible to easily make a scrolling background. Such a demo is included in the LOLGraphics editor as an example program. However for this, you need an image that is larger than the screen so that you will actually have to scroll through. It's possible to load that image from a file (you will learn how in this chapter), however the example program uses a very large image that is included in LOLGraphics. Can be drawn using the command `PLZ DRAW BACKGROUND`.

Just don't forget that if you want to move left, you are actually moving the image right. If you want to move right, you are actually moving the image left. If you want to move up, you are actually moving the image down, and if you want to move down, you are actually moving the image up.

Scrolling background source code

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A ONE BYTE DAT IZ CALLED BUTTON

PLZ SET TWO BYTE X 0
PLZ SET TWO BYTE Y 0
PLZ DRAW BACKGROUND

FOREVER RUN CHECK

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED CHECK
PLZ SET ONE BYTE BUTTON -1
PLZ READ ONE BYTE BUTTON

SWITCH [BUTTON]
CASE 8 UP
CASE 2 DOWN
CASE 4 LEFT
CASE 6 RIGHT
CASE 86,68,9 UP_RIGHT
CASE 48,84,7 UP_LEFT
CASE 42,24,1 DOWN_LEFT
CASE 62,26,3 DOWN_RIGHT

IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE X X+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE X X-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP_RIGHT
PLZ SET TWO BYTE X X-10
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP_LEFT
PLZ SET TWO BYTE X X+10
PLZ SET TWO BYTE Y Y+10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM
```

```
IM IN UR SUBPROGRAM DAT IZ KALLED DOWN_RIGHT
PLZ SET TWO BYTE X X-10
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN_LEFT
PLZ SET TWO BYTE X X+10
PLZ SET TWO BYTE Y Y-10
PLZ DELIVR MAH CHEEZBURGERS 2 X Y
PLZ CLEAR TEH SCREEN
PLZ DRAW BACKGROUND
IM OUTTA UR SUBPROGRAM
```

Drawing lines & shapes

To draw a line, write `PLZ DRAW LINE <x1> <y1> <x2> <y2>`. The coordinates x1, y1, x2, y2 can be either numbers or 2-byte variables. You can also draw in custom colors using `PLZ CHANGE PAINT BRUSH <color>`. Like with custom locations for cheeseburgers, custom paint colors will not affect what is already on the screen, only what will be drawn after that.

You can draw a rectangle using the command `PLZ DRAW RECT <x1> <y1> <width> <height>`, fill a rectangle using the command `PLZ FILL RECT <x1> <y1> <width> <height>`. In addition, you can draw an ellipse using the command `PLZ DRAW ELLIPSE <x1> <y1> <width> <height>`, and fill using the command `PLZ FILL ELLIPSE <x1> <y1> <width> <height>`. You can also draw a custom polygon using the command `PLZ DRAW POLY <points>`. Replace <points> with an even number of arguments that is greater or equal to 6 which represent x,y coordinates of at least 3 points.

All the above commands accept as arguments numbers or 2 byte variables. Also if you want to draw a shape with an outline, first call the fill command, them change the paint color, and only then call the draw command.

Also, it's possible to fill the entire screen by the color of the paintbrush using the command `PLZ FILL TEH SCREEN`.

Loading images from file

To load an image use the command `PLZ LOAD IMAGE EXAMPLE EXAMPLE.PNG`. If you want to draw example.png on the screen, type `PLZ DRAW IMAGE EXAMPLE`. This will draw the image on the location set for cheeseburgers.


IM IN UR FOLDUR
KERUPTIN YR FYLEZ

Making a simple game in LOLGraphics

Introduction

In this tutorial we will make a simple WASD controlled game where you control a square. If you fall in the lava pool, you lose, and if you get to the purple carpet, you will be asked a multiple-choice trivia question (who was the first president of the United States). If you answer it correctly, you win. If you answer it wrong, you lose. Simple enough.

Since the code will be written in parts, this chapter will follow the following conventions: new lines that didn't exist in the previous version will be colored by green, lines that should be removed will be colored by red, and lines that should stay as they are will be colored by blue.

Drawing the field

You will start of with an empty program. Let's start by drawing the field. Yeh background will be green like grass, there will be one red square representing a pool of lava, and one magenta square.

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DRAW_FIELD

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
```

```
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
IM OUTTA UR SUBPROGRAM
```

Notice that the delay was set to 1. This is the lowest possible and was done on purpose so that the game will run as smoothly as possible.

Drawing the player

The player will be an orange square that can move around the screen.

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DRAW_FIELD
PLZ RUN SUBPROGRAM DEFALT_VARIABLES

IM OUTTA UR CODE

 IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

 IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
IM OUTTA UR SUBPROGRAM
```

Notice how the drawing happens in the same subprogram where the default variables are set. This will change.

Player movement

As it is, we don’t have much of a game. Let’s implement player movement:

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DEFALT_VARIABLES
PLZ RUN SUBPROGRAM DRAW_FIELD
PLZ RUN SUBPROGRAM DEFALT_VARIABLES
FOREVER RUN MOVE

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A TWO BYTE DAT IZ CALLED BUTTON
```

```
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ SET TWO BYTE BUTTON -1
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED MOVE
PLZ READ CHAR BUTTON
SWITCH [[BUTTON]]
CASE 56,87,119 UP
CASE 52,65,97 LEFT
CASE 50,83,115 DOWN
CASE 54,68,100 RIGHT
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM
```

Notice that the player drawing routine was moved to the field, and that the program now sets default values before it starts drawing. Also in the switch/case statement, it compares the variables to the ASCII values of WASD and 8462 keys.

Detecting Lava

Now we have completed the movement, but you can walk wherever you want including on top of lava. Let's fix that. Let's add code that detects when you fell in lava and end the game.

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DEFALT_VARIABLES
PLZ RUN SUBPROGRAM DRAW_FIELD
FOREVER RUN MOVE

IM OUTTA UR CODE
```

```
IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A TWO BYTE DAT IZ CALLED BUTTON
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ SET TWO BYTE BUTTON -1
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED MOVE
PLZ READ CHAR BUTTON
SWITCH [[BUTTON]]
CASE 56,87,119 UP
CASE 52,65,97 LEFT
CASE 50,83,115 DOWN
CASE 54,68,100 RIGHT
PLZ RUN SUBPROGRAM CKECK_LAVA_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED FELL_IN_LAVA
PLZ PRINT TEXT YOU FELL IN LAVA
IM OUTTA UR CODE
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_X
SWITCH [[X]]
CASE 80,90,100,110,120,130,140 CKECK_LAVA_Y
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_Y
SWITCH [[Y]]
CASE 80,90,100,110,120,130,140 FELL_IN_LAVA
IM OUTTA UR SUBPROGRAM
```

Now if you fall in the lava, you get a message informing you about this, and the game ends. Notice that because the player starts at coordinates (200,150) and moves at a speed of 10 pixels/iteration, both the x and y values will always be numbers that can evenly be divided by 10.

Detecting Carpet

Now we can detect when the player enters lava, let's also detect when the player steps on the carpet.

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DEFALT_VARIABLES
PLZ RUN SUBPROGRAM DRAW_FIELD
FOREVER RUN MOVE

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A TWO BYTE DAT IZ CALLED BUTTON
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ SET TWO BYTE BUTTON -1
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED MOVE
PLZ READ CHAR BUTTON
SWITCH [[BUTTON]]
CASE 56,87,119 UP
CASE 52,65,97 LEFT
CASE 50,83,115 DOWN
CASE 54,68,100 RIGHT
PLZ RUN SUBPROGRAM CKECK_LAVA_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED FELL_IN_LAVA
PLZ PRINT TEXT YOU FELL IN LAVA
IM OUTTA UR CODE
IM OUTTA UR SUBPROGRAM
```

```
IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_X
SWITCH [[X]]
CASE 80,90,100,110,120,130,140 CKECK_LAVA_Y
PLZ RUN SUBPROGRAM CKECK_CARPET_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_Y
SWITCH [[Y]]
CASE 80,90,100,110,120,130,140 FELL_IN_LAVA
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED ON_CARPET
PLZ PRINT TEXT YOU WIN
IM OUTTA UR CODE
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_X
SWITCH [[X]]
CASE 280,290,300,310,320,330,340 CKECK_CARPET_Y
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_Y
SWITCH [[Y]]
CASE 180,190,200,210,220,230,240 ON_CARPET
IM OUTTA UR SUBPROGRAM
```

Trivia question

Now when you step on the carpet, you win instantly. Let’s make it so that you will get a trivia question and only win if you answered correctly.

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DEFALT_VARIABLES
PLZ RUN SUBPROGRAM DRAW_FIELD
FOREVER RUN MOVE

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A TWO BYTE DAT IZ CALLED BUTTON
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ SET TWO BYTE BUTTON -1
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50
PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED MOVE
PLZ READ CHAR BUTTON
SWITCH [[BUTTON]]
CASE 56,87,119 UP
CASE 52,65,97 LEFT
CASE 50,83,115 DOWN
CASE 54,68,100 RIGHT
PLZ RUN SUBPROGRAM CKECK_LAVA_X
IM OUTTA UR SUBPROGRAM
```

```
IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED FELL_IN_LAVA
PLZ PRINT TEXT YOU FELL IN LAVA
PLZ PRINT TEXT
PLZ PRINT TEXT YOU LOSE
IM OUTTA UR CODE
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_X
SWITCH [[X]]
CASE 80,90,100,110,120,130,140 CKECK_LAVA_Y
PLZ RUN SUBPROGRAM CKECK_CARPET_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_Y
SWITCH [[Y]]
CASE 80,90,100,110,120,130,140 FELL_IN_LAVA
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED ON_CARPET
PLZ PRINT TEXT WHO WAS THE FIRST
PLZ PRINT TEXT WHO PRESIDENT OF THE
PLZ PRINT TEXT UNITED STATES?
PLZ PRINT TEXT
PLZ PRINT TEXT 1) RICHARD NIXON
PLZ PRINT TEXT 2) GEORGE WASHINGTON
PLZ PRINT TEXT 3) DONALD TRUMP
PLZ PRINT TEXT 4) JOHN ADAMS

I HAS A ONE BYTE DAT IZ CALLED ANSWER
PLZ ASK TEH USR 2 GIMME A ONE BYTE ANSWER
PLZ CLEAR TEH CONSOLE
SWITCH [ANSWER]
LABELCASE 2 WIN
PLZ PRINT TEXT YOU LOSE!
PLZ GOTO LABEL END
DIS IZ MY LABEL! IT IZ KALLED WIN
PLZ PRINT TEXT YOU WIN!
DIS IZ MY LABEL! IT IZ KALLED END
IM OUTTA UR CODE
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_X
SWITCH [[X]]
CASE 280,290,300,310,320,330,340 CKECK_CARPET_Y
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_Y
SWITCH [[Y]]
CASE 180,190,200,210,220,230,240 ON_CARPET
IM OUTTA UR SUBPROGRAM
```

Now when you step on the carpet, you will get asked who was the first president of the United States. Also, when you fall im lava, the text “You lose” is displayed in addition to “You fell in lava”. This “game” can be considered complete, however there’s one more thing we are going to add.

Slight improvement

You would expect that after the player falls in lava, he would not be drawn on the screen. However this is not what happens in the game as of now. Let’s fix this. There are multiple ways to do so:

1. One way, is to set the x and y values of the player to something out of bounds and then redraw the screen. This will cause the player to be “drawn” but because it will be out of the screen you won’t see him. Feel free to use this way, but this is not what we are going to use in this book.
2. The second way is it to create a one-byte variable called IN_LAVA, set it to 0, and when the player falls in lava, set it to 1. The drawing routine will check the value of this variable and set if it’s equal to 1, the player won’t be drawn. Feel free to use it in your code, but this isn’t what we will be doing in this book.
3. The third way would be to set the value of x or y to a number like -1 and then in the drawing routine to check if x==-1, and if yes then the player won’t be drawn. You can use any number that can’t be evenly divided by 10 and it won’t have any effect on how the code runs because the player moves at a speed of 10 pixel/iteration and starts with x,y coordinates that can be evenly divided by 10. This is what we will do.

So here’s the improved code:

```
HAI 3.4 0 1
IM IN UR CODE EXECUTIN UR KOMANDZ

PLZ RUN SUBPROGRAM DEFALT_VARIABLES
PLZ RUN SUBPROGRAM DRAW_FIELD
FOREVER RUN MOVE

IM OUTTA UR CODE

IM IN UR SUBPROGRAM DAT IZ KALLED DEFALT_VARIABLES
I HAS A TWO BYTE DAT IZ CALLED X
I HAS A TWO BYTE DAT IZ CALLED Y
I HAS A TWO BYTE DAT IZ CALLED BUTTON
PLZ SET TWO BYTE X 200
PLZ SET TWO BYTE Y 150
PLZ SET TWO BYTE BUTTON -1
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DRAW_FIELD
PLZ CHANGE PAINT BRUSH 0 154 23
PLZ FILL RECT 0 0 630 430
PLZ CHANGE PAINT BRUSH RED
PLZ FILL RECT 100 100 50 50
PLZ CHANGE PAINT BRUSH MAGENTA
PLZ FILL RECT 300 200 50 50

SWITCH [[X]]
LABELCASE -1 END

PLZ CHANGE PAINT BRUSH ORANGE
PLZ FILL RECT X Y 25 25
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED MOVE
PLZ READ CHAR BUTTON
SWITCH [[BUTTON]]
CASE 56,87,119 UP
CASE 52,65,97 LEFT
```

```
CASE 50,83,115 DOWN
CASE 54,68,100 RIGHT
PLZ RUN SUBPROGRAM CKECK_LAVA_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED UP
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED DOWN
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE Y Y+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED LEFT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X-10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED RIGHT
PLZ SET TWO BYTE BUTTON -1
PLZ SET TWO BYTE X X+10
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED FELL_IN_LAVA
PLZ PRINT TEXT YOU FELL IN LAVA
PLZ PRINT TEXT
PLZ PRINT TEXT YOU LOSE
PLZ SET TWO BYTE X -1
PLZ RUN SUBPROGRAM DRAW_FIELD
IM OUTTA UR CODE

IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_X
SWITCH [[X]]
CASE 80,90,100,110,120,130,140 CKECK_LAVA_Y
PLZ RUN SUBPROGRAM CKECK_CARPET_X
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_LAVA_Y
SWITCH [[Y]]
CASE 80,90,100,110,120,130,140 FELL_IN_LAVA
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED ON_CARPET
PLZ PRINT TEXT WHO WAS THE FIRST
PLZ PRINT TEXT WHO PRESIDENT OF THE
PLZ PRINT TEXT UNITED STATES?
PLZ PRINT TEXT
PLZ PRINT TEXT 1) RICHARD NIXON
PLZ PRINT TEXT 2) GEORGE WASHINGTON
PLZ PRINT TEXT 3) DONALD TRUMP
PLZ PRINT TEXT 4) JOHN ADAMS

I HAS A ONE BYTE DAT IZ CALLED ANSWER
PLZ ASK TEH USR 2 GIMME A ONE BYTE ANSWER
PLZ CLEAR TEH CONSOLE
SWITCH [ANSWER]
LABELCASE 2 WIN
PLZ PRINT TEXT YOU LOSE!
PLZ GOTO LABEL END
DIS IZ MY LABEL! IT IZ KALLED WIN
PLZ PRINT TEXT YOU WIN!
DIS IZ MY LABEL! IT IZ KALLED END
IM OUTTA UR CODE

IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_X
SWITCH [[X]]
CASE 280,290,300,310,320,330,340 CKECK_CARPET_Y
```

```
IM OUTTA UR SUBPROGRAM

IM IN UR SUBPROGRAM DAT IZ KALLED CKECK_CARPET_Y
SWITCH [[Y]]
CASE 180,190,200,210,220,230,240 ON_CARPET
IM OUTTA UR SUBPROGRAM
```

Notice that the label “END” is the same one in the carpet subprogram. This doesn’t matter since after it there’s `IM OUTTA UR CODE` which ends the program.

LOLGraphics Cheat Sheet

*Note that this page only explains what the commands do in version 3.4 and some of them have slightly changed functionality or were only added in later versions. For more info, please see the itch.io page.

Command	What it does
PLZ TYPE TEXT <string>	prints the string to the console, doesn't go to the next line.
PLZ PRINT TEXT <string>	prints the string to the console, goes to the next line after that.
PLZ TYPE ONE BYTE/TWO BYTE/FOUR BYTE/EIGHT BYTE <index/variable>	prints the value in the given index or variable in the given segment. Doesn't go to the next line.
PLZ PRINT ONE BYTE/TWO BYTE/FOUR BYTE/EIGHT BYTE <index/variable>	prints the value in the given index or variable in the given segment. Goes to the next line after that.
PLZ ADD A SPACE	Adds a space to the terminal.
I CAN HAS A CHEEZBURGER?	draws a cheeseburger on the graphics area. You can draw multiple cheeseburgers on the screen, and choose their location using the command `PLZ DELIVR MAH CHEEZBURGERS 2`
PLZ DELIVR MAH CHEEZBURGERS 2 <x> <y>	sets the position of cheeseburgers drawn after this command is run. Does not affect existing cheeseburgers already on the screen.
PLZ WAIT 4 DA USR 2 REACT	activates the input button, and waits for the user to press it. Only then will the program continue to the next line. Useful for debugging.
PLZ CLEAR TEH SCREEN	clears the graphics area.
PLZ CLEAR TEH CONSOLE	clears the console.
I HAS A ONE BYTE/TWO BYTE/FOUR BYTE/EIGHT BYTE DAT IZ CALLED <name>	defines a variable of the given size with the given name. What it actually does is giving a label to the first available address in the specified segment. If you define multiple variables with the same name, the code will work but will always access the first one.
PLZ SET ONE/TWO/FOUR/EIGHT BYTE <address/variable> <number/variable>	sets the specified address or variable to the specified variable. All variables must be in the same segment. Also you can specify arithmetic operations, but there is a limitation. You can't use more than one type of operation per line. `PLZ SET EIGHT BYTE B C+5+A` is valid but `...C-5+A` is not.
PLZ GIMME A RANDOM ONE/TWO/FOUR/EIGHT BYTE <address/variable>	sets the specified address or variable to a random number.
PLZ ASK TEH USR 2 GIMME A ONE/TWO/FOUR/EIGHT BYTE <address/variable>	halts the program until the user types a number and presses the input button. Then, the number is stored in the specified variable, and removed from the input field.
IM IN UR SUBPROGRAM DAT IZ KALLED <name>	write this line at the beginning of every subprogram. By design, if the code is written correctly, the interpreter will never try to run this line, however if you put it in a place where it will be reached, you will see a message saying that the interpreter doesn't know what to do.
PLZ RUN	runs the specified subprogram.

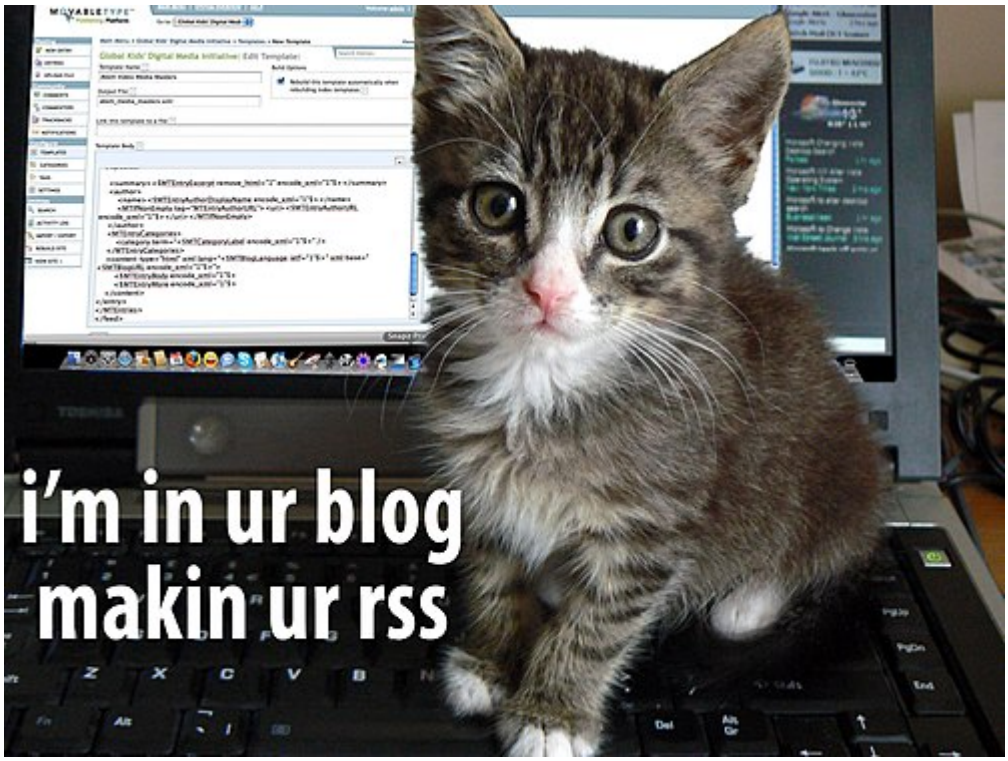
SUBPROGRAM <name>	
IM OUTTA UR SUBPROGRAM	ends the current subprogram, goes back to run the line after the one that called the subprogram.
PLZ ASK CEILIN KAT 2 CHEK IZ <condition>	checks if the specified condition is true. If it is, saves true in the flag. If it's false, saves false. Example: `PLZ ASK CEILIN KAT 2 CHEK IZ [I]<[MAX]`. Allowed operations: < == >.
IF CEILIN KAT IZ NODDING PLZ RUN <subprogram>	runs the specified subprogram if the value of the flag is true.
ELSE PLZ RUN <program>	runs the specified subprogram if the value of the flag is false. A valid command, independent of the if command.
PLZ ASK CEILIN CAT 2 NOD	sets the value of the flag to true.
PLZ ASK CEILIN CAT 2 STOP NODING	sets the value of the flag to false
FOREVER RUN <subprogram>	runs the specified subprogram in an infinite loop
WHILE CEILIN CAT IZ NODDIN PLZ RUN <subprogram>	runs the subprogram while the value of the flag is true. Don't forget to add a condition at the end of the subprogram in order to update the flag otherwise you will get an infinite loop.
PLZ SET DELAY <number>	changes the amount of time that passes in between every two commands.
PLZ READ ONE/TWO/FOUR/EIGHT BYTE <address/variable>	doesn't wait for the user and reads the number in the input field. Saves the number of the specified variable or address. Also clears it.
SWITCH <variable/address>	copies the value of the variable or address to an 8 bit variable in the memory. Unlike most programming languages, this is a command and not a structure.
CASE <number/a list of numbers separated by commas without spaces> <subprogram>	checks if the value of the variable used in switch is equal to the specified number or one of the specified numbers. If yes, runs the specified subprogram.
LABELCASSE <number/numbers seperated by commas without spaces> <label>	like case, only jumps to a label instead of running a subprogram.
PLZ CHANGE TEXT COLOR <color>	changes the color of the text in the console. Can get the name of the color (black, blue, cyan, dark gray, gray, green, light gray, magenta, orange, pink, red, white, or yellow): `PLZ CHANGE TEXT COLOR CYAN` Can also accept an rgb value: `... 100 100 100`. In addition can also accept the word random that will change the text color to a random number.
PLZ LOAD IMAGE <image name in the code> <file name>	loads an image. Example: `PLZ LOAD IMAGE EXAMPLE EXAMPLE.PNG`
PLZ DRAW IMAGE <image name>	draws the specified image on the screen on the location chosen for drawing cheeseburgers.
PLZ DRAW LINE <x1> <y1> <x2> <y2>	draws a line from point (x1,y1) to (x2,y2). x1, y1, x2, and y2 can be numbers or two byte variables.
PLZ CHANGE PAINT BRUSH <color>	changes the color of lines drawn in the future. Does not affect existing lines. The color can be specified either by its name (list of supported colors in `PLZ CHANGE TEXT COLOR <color>`), by its rgb value, or a random color by typing the word RANDOM.

PLZ GIMME A RANDOM ONE/TWO/FOUR/EIGHT BYTE IN RANGE <address/variable> <min> <max>	generates a random number in the given range and saves it in the specified address or variable. min and max can be just numbers, not variables.
PLZ DRAW RECT <x1> <y1> <width> <height>	draws the outline of a rectangle with the top left corner in the specified coordinates (x1,y1) and with the specified width and height. You can enter as parameters numbers or two byte variables.
PLZ FILL RECT <x1> <y1> <width> <height>	draws and fills a rectangle with the top left corner in the specified coordinates (x1,y1) and with the specified width and height. You can enter as parameters numbers or two byte variables. If you want to draw a rectangle with an outline in another color, first write `PLZ FILL RECT` and only then `PLZ DRAW RECT`
PLZ DRAW ELLIPSE <x1> <y1> <width> <height>	draws the outline of an ellipse with the top left corner in the specified coordinates (x1,y1) and with the specified width and height. You can enter as parameters numbers or two byte variables.
PLZ FILL ELLIPSE <x1> <y1> <width> <height>	draws and fills a ellipse with the top left corner in the specified coordinates (x1,y1) and with the specified width and height. You can enter as parameters numbers or two byte variables.
PLZ FILL TEH SCREEN	colors the entire screen with the color of the paint brush. Can be changed using the command `PLZ CHANGE PAINT BRUSH`
PLZ DRAW POLY <points>	draws a polygon through the specified points. Must be at least 3 points (6 numbers) and an even number of numbers (x,y coordinates). `PLZ DRAW POLY 10 10 150 200 100 20` is a valid command while `PLZ DRAW POLY 10 10 150 200 100 20 5` and `PLZ DRAW POLY 10 10 150 200` are not.
DIS IZ MY LABEL! IT IZ KALLED <name>	a label
PLZ GOTO LABEL <label>	jumps to the specified label.
PLZ READ CHAR <address/variable>	reads the character that was entered and saves it's unicode value in the specified two bytes variable, or the specified address in the two bytes segment. Only supports two bytes values. If more than one character is provided, it will only put attention to the first character.
PLZ DRAW BACKGROUND	draws a very large image that can't fit the screen. Used in the scrolling demo that is included in the editor.

External links

Simply click on the link you want to enter, or if you are reading a printed version, type it in your browser of choice:

1. https://bloody-wall-software.itch.io/lolgraphics (https://bloody-wall-software.itch.io/lolgraphics) - link to download from itch.io
2. https://esolangs.org/wiki/LOLGraphics (https://esolangs.org/wiki/LOLGraphics) - entry in esolangs.org
3. https://pwwe47.wixsite.com/bloodywallsoftware/lolgraphics (https://pwwe47.wixsite.com/bloodywallsoftware/lolgraphics) - entry in the website of Bloody Wall Software



Authors & Contributors

If you have contributed in any way to this book, please sign here by typing ~~~.

1. Gifnk dlm 2020 If only Middle English Wikipedia could be saved(talk)
2. Mbrickn - Various small edits.